Applied Machine Learning Coursework 2

Fine-grained image classification: Dog Breeds

Authors:
Rongjian Huang, Nithi Lertpongpaism, Yonghan Pan, Wenjia Wang, Charles Wimmer, Minjie Xu
Project Supervisor:
Stefano Zappala

Contents

Introduction

Our task is to use the given images dataset to build a neutral networks training model, which can predict the breed of dogs in the test image dataset. Although the task sounds simple, this task needs the good coordination of proper data pre-processing and appropriate training model.

The main body of this report covers 4 parts: descriptive analysis of the task and dataset, model building, analysis of implementation result and conclusion part. Firstly, we analyzed the contents of the dataset. On the basis of that is data preprocessing made. Then we read some literature about image recognition and categorization; decided to use VGG16 and Xception for categorization model building. After model building, we draw some graphs of the results of implementation and accuracy of the model. Some information of the research about CNN and related work is given as a part of literature review in this report. Last but not least, we also wrote some insights of the possible improvement for the categorization work in the future.

Descriptive Analysis

For this project we used the Stanford Dog Dataset. The dataset is composed of photos of a total of 20,580 JPEG images of 120 breeds of dogs from all around the world, and annotation data regarding them. The dataset can be freely downloaded from the Stanford University website at http://vision.stanford.edu/aditya86/ImageNetDogs/ .

The useful parts of the dataset as used for this project are the following:

1. The images themselves

2. XML annotation files, one for each image, containing labels and bounding boxes of the dog(s) in the image

3. "List" files in Matlab format, where each line has: file path to an image, file path to annotation file, and a numerical label. There is a full list, and two lists split as test and train sets

To begin with, we set up some blank arrays to illustrate some basic information of the dataset. Then a function is defined to load the list files and iterate over every line in a loop. Using this we were able to count the total number of images and confirm that every image has an annotation file paired with it. The annotation files let us count the total number of dogs, as each dog is registered as a separate object in the XML. We can also load every image using the Pillow library and calculate the size of the data we are working with.

It was found that there are 22,126 dogs in total. The average area of original images is about 181468.574 px^2 and the average area of bounding boxes is about 91938.762 px^2. We divided the average area of bounding boxes by the average area of original images and got the percentage of the overall reduction in size which is about 50.7%. Thus by utilising the provided boxes we would reduce the amount of data we would have to process by about 50%. In addition, a table can be compiled to show the number of images of each breed. The head and tail are shown in Table 1.

From the table, the “Afghan Hound” is the most numerous with 287 images. Meanwhile, “Redbone” has the minimum amount with 151 images. There is a mean average of 184.4 images. This may mean we don’t have as much data to train a Convolutional Neural Net (CNN) classifier as would be optimal. Therefore, we tried to perform some data augmentation, and split the original dataset into train/development/test sets at proportions of 8:1:1, in order to keep the majority of the data for good training. This leaves the numbers of pictures in every set as: 16464, 2058, and 2058, respectively. We used the development set to adjust the parameters of models, and the test set for the final result, which will be described more detailed in the experimental setting part.

	Breed	Count
0	Chihuahua	158
1	Japanese_spaniel	202
2	Maltese_dog	264
3	Pekinese	152
4	Shih-Tzu	233
...	...	...
115	standard_poodle	173
116	Mexican_hairless	162
117	dingo	170
118	dhole	179
119	African_hunting_dog	213

120 rows × 2 columns

```
max count: 287 | min count: 151 | average: 184.4
```

Table 1

It was found that the filenames and other fields provided within the annotation XML files were not always present for unknown reasons, sometimes replaced by “%s”, presumably due to an error with whatever software was used to generate them. This only occured with information also available in the Matlab list files, so it wasn’t an issue. Fortunately, all of the object details in the annotations (e.g. bounding box coordinates) were proper and intact, so we did not need to make any adjustments there.

Finally, whilst the dataset does contain train and test splits in the form of two list files, these are not useful to us as they are split roughly 60:40 and do not contain a separate development/validation set, thus these files were ignored henceforth.

Methodology

As our team hasn't got any computer with a strong GPU, we used Colab to do this project with its GPU.

To set up the environment, we first mount google drive to Colab, this enables us to upload the compress dog images from google drive to Colab. Then we wrote a function to decompress these images in the Colab system.

For the preprocessing part, we cropped the images into dog-only parts and we also keep the original images, so we have two versions to compare to get the best performance on whether to crop or not. Then we use 'split-folders' to help us split the images in train, validation and test set in 8:1:1 ratio. Also we use data augmentation to better address the overfitting problem. To be specific, we rotate, shift, shear, zoom and also flip the images. To avoid we may run out of memory, a data generator was used to create batch data from images.

For the implementation part, we use several models. First we have tried to use a model created by ourself, we used relu as our activation function, sparse categorical cross entropy as our loss function. The structure of the network (figure 1) is some convolutional and pooling layers on top to extract features, a global average pooling to replace the fully connected layer and a dense softmax layer to come out the result label.

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 148, 148, 64)	1792
max_pooling2d (MaxPooling2D)	(None, 74, 74, 64)	0
conv2d_5 (Conv2D)	(None, 72, 72, 128)	73856
max_pooling2d_1 (MaxPooling2	(None, 36, 36, 128)	0
conv2d_6 (Conv2D)	(None, 34, 34, 256)	295168
max_pooling2d_2 (MaxPooling2	(None, 17, 17, 256)	0
global_average_pooling2d (Gl	(None, 256)	0
dense (Dense)	(None, 120)	30840

Total params: 401,656
Trainable params: 401,656
Non-trainable params: 0

Figure 1

But we find out the result accuracy is only 20% which is way too low, we use an idea of transfer learning, which means we use some pre-trained and mature network that works well in this field.

So that we only need to fine tune some layers rather than all layers. We first tried VGG16 in the Imagenet library, and our validation accuracy increased to 60%(Figure 2). This part we removed from the code.

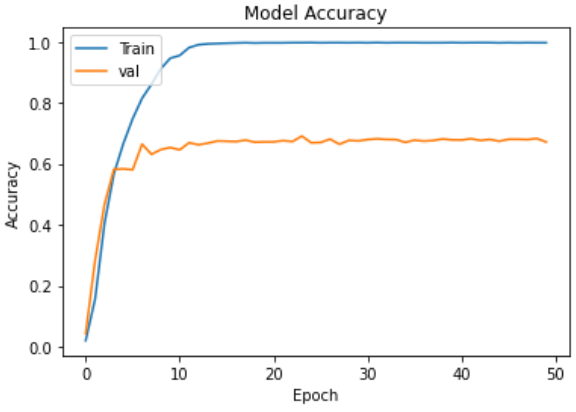

Figure 2

Then we tried the Xception network to train the data sets, and our validation accuracy increased to 86% (Figure 3). Obviously, the model is overfitting because of the high training accuracy with 99%. If we want to increase the accuracy of the validation set, the problem of overfitting has to be solved.

```
Epoch 12/50
372/372 [==============================] - 173s 464ms/step - loss: 0.1645 - accuracy: 0.9689 - val_loss: 0.4490 - val_accuracy: 0.8615
Epoch 13/50
372/372 [==============================] - 173s 464ms/step - loss: 0.1650 - accuracy: 0.9655 - val_loss: 0.5180 - val_accuracy: 0.8370
Epoch 14/50
372/372 [==============================] - 173s 465ms/step - loss: 0.1223 - accuracy: 0.9793 - val_loss: 0.4895 - val_accuracy: 0.8463
Epoch 15/50
372/372 [==============================] - 173s 464ms/step - loss: 0.1232 - accuracy: 0.9761 - val_loss: 0.4832 - val_accuracy: 0.8460
Epoch 16/50
372/372 [==============================] - 173s 464ms/step - loss: 0.1006 - accuracy: 0.9839 - val_loss: 0.5157 - val_accuracy: 0.8402
Epoch 17/50
372/372 [==============================] - 173s 464ms/step - loss: 0.0854 - accuracy: 0.9874 - val_loss: 0.4755 - val_accuracy: 0.8560
Epoch 18/50
372/372 [==============================] - 173s 464ms/step - loss: 0.0779 - accuracy: 0.9892 - val_loss: 0.5365 - val_accuracy: 0.8433
Epoch 19/50
372/372 [==============================] - 173s 464ms/step - loss: 0.0620 - accuracy: 0.9938 - val_loss: 0.5419 - val_accuracy: 0.8300
Epoch 20/50
```

Figure 3

We finished choosing our initial Xception model and started to improve the model by tuning parameters.

Experimental Setting

1. Solving the problem of overfitting

a) Data augmentation

Data augmentation is to solve the problem of overfitting by increasing the number of training images. The model will not view the exact same image twice during training. This allows the model to observe more data and thus have better generalization capabilities.

We generate augmented data before training the classifier. ImageDataGenerator() is used to make basic transformations for all images in the training set. All these images are fed into the net at training time. (Perez and Wang, 2020) Figure 4 shows four types of transformations for one image.

Data Augmentation result in the decrease of accuracy of the training set but help solve overfitting and improve the accuracy of developments set on a small scale.(Table 1)

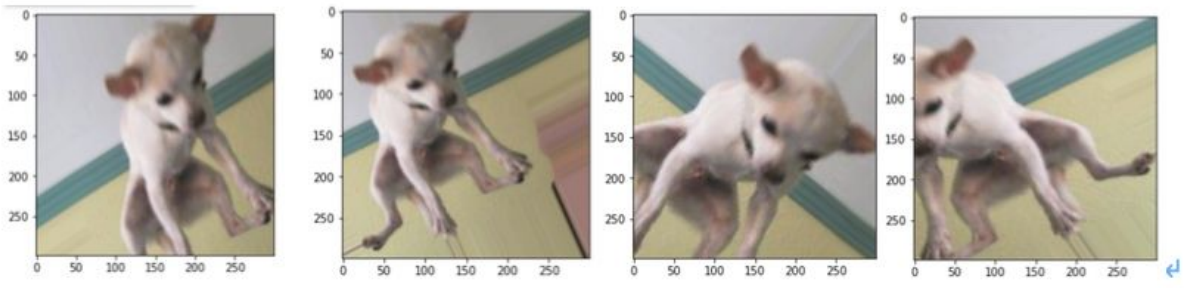

Figure 4

Solutions	Train_Acc	Val_Acc	After cropped Train_Acc	After cropped Val_Acc
Xception	99%	86%	99%	88%
Data Augmentation + Xception	94%	70%	96%	89%

Table 1

b) Dropout

Dropout is the most effective way of regularization to solve overfitting. It can discard some features randomly of the upper layer. The dropout rate is set between 0.2 and 0.5. We set it 0.3 after a Dense layer in the model. This way decreases the accuracy of the training set but dropout() is useless for development and testing sets.

c) Feature extraction

We use the convolutional base of the model to train a new classifier. The convolutional base can obtain the feature of objective location. The generality (and reusability) of the representation extracted by a convolutional layer depends on the depth of the layer in the model. The layer closer to the bottom in the model extracts local, highly versatile feature maps (such as visual edges, colors, and textures), while the layer closer to the top extracts more abstract concepts (such as "dog ears" or "dog eyes").Thus, except for the convolutional base from the Xception, we also try the convolutional base from the InceptionResnet who has the deepest layers.

Two ways to extract features:

- Feature extraction without data augmentation
- Feature extraction with data augmentation

If the features are extracted without data augmentation, the training efficiency can be increased compared with feature extraction with data augmentation. More time is used in feature extraction than fitting the model. If we pay more attention to the speed of improving the model result, we can apply it.

Two kinds of convolutional bases to extract features:

- Convolutional base from the Xception
- Convolutional base from the InceptionResnet

Feature extraction helps solve the problem of overfitting and decreases the accuracy of the training set. It helps increase the accuracy of the development set. We can see that extracting features by the convolutional base of InceptionResnet with data augmentation and dropout can result in the highest accuracy.(Table 2)

Solutions	Train Acc	Val Acc	After cropped Train Acc	After cropped Val Acc
a. Xception	99%	86%	99%	88%
b. Feature Extraction by Xception	99%	60%	RAM blew up	RAM blew up
c. Feature Extraction by InceptionResnet	99%	66%	RAM blew up	RAM blew up
d. Data Augmentation+Feature Extraction by Xception+dropout()	50%	57%	87%	91%
e. Data Augmentation Feature Extraction by InceptionResnet+dropout()	54%	60%	84%	93%

Table 2

2. Class mode and Loss function

We choose the sparse categorical cross entropy as the loss function because we have multiple classes and each sample belongs exactly to one class. We chose the categorical cross entropy as the loss function by mistake in the beginning, which resulted in the low accuracy. Then we find categorical cross entropy is chosen when one sample belongs to multiple classes. The class mode"sparse"is used here to get integer labels , also matching the loss function.

3. Optimizer comparison

Common choices for the optimizer are either a fixed value, a predetermined schedule or an adaptive scheme based on curvature or gradient information (Keskar and Saon, 2015). SGD and Adam are the main optimizers compared in the model.

SGD is our first choice of the optimizer with default parameters as an attempt. Surprisingly, it results in a good result. However, the accuracy fluctuates around one value because of the fixed but too large learning rate. Thus, the callback function of reducing learning rate when the accuracy doesn't change is added. Then the accuracy is improved on a small scale.

Adam is an adaptive scheme based on momentum, which is stronger than SGD. The Parameters ß1 is used in the first-order momentum and ß2 is used in the second-order momentum. The problem of choosing parameters has been investigated by the

optimization community with many options available which ensure global convergence under regularity conditions. The recommended parameters:

```
optimizer=keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999,
epsilon=None, decay=0.0, amsgrad=False), metrics = ["accuracy"])
```

We can get the result that the optimizer SGD with the callback function of reducing the learning rate can get a higher accuracy.(Table 3)

Solutions	Train Acc After cropped	Dev Acc After cropped
Data Augmentation Feature Extraction by InceptionResnet+dropout(). --using SGD	84%	93%
Data Augmentation+Feature Extraction by Inception Resnet+dropout() --using Adam	47%	80%

Table 3

4. Batch size

The choice of batch size should be between 2 and 32m because mini-batch can update parameters many times in one epoch to speed up convergence. (Masters and Luschi, 2020) Another reason is that we used the batch of data instead of the full data can Escaping From Saddle Points. (Ge, Huang, Jin and Yuan, 2020) We set batch size 20 here.

Result

As a result, it was found that the best accuracy is produced by the model InceptionResnet, using SGD as the optimizer with the callback function of reducing the learning rate after solving the problem of overfitting by the data augmentation, feature extraction and dropout. This produced an accuracy of 93%.

Error Analysis

In the first version, we use only the self-create model which has only 8 layers and 400,000 parameters. The accuracy is 20% and it is unacceptable. After consideration, we think this problem is cause by two aspect:

1. We didn’t crop the image, so the noisy background makes it hard to learn the dogs’ features.

2. The model is not deep enough to learn the pattern of different dogs.

So we made some improvement respectively. First we wrote a program to crop the image to leave only dogs in the images, so that the model is able to learn only the features of the dog. Second, rather than using a self-created model, we used some existing and mature models that work well in image recognition fields such as Xception, Vgg16 and Inception. It turns out that our direction is right.

We now have 99% of train accuracy but 60% of validation accuracy. Here comes to another problem, overfitting. To make the model generalize well in the test set, we apply drop out to the model and apply data augmentation to our images. The validation accuracy has risen to 93%, though there is still a little bit of overfitting.

We also face the library version problem. Since we have different work allocation for every group member, our project is combined with different members' code. So here comes the problem when the code works well in one's laptop but fails in others. So we set up the standard version of the library we commonly used in this project to avoid this problem.

Literature Review

Firstly, we read some literature reviews about Object Detection and introduction to some algorithms of Fine-Grained image categorization. In recent years, one of the most inspirable algorithms is YOLO. Because this algorithm could train the dataset for image categorization more efficiently as the basic part of a final model. P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan (2010) created a classifier model using a bounding box that runs at evenly step by step over the entire image. But Joseph R., Santosh D., Ross G., Ali F. (2016) trained the YOLO on full images and that could optimize detection performance directly.

There are several pieces of previous research that study image recognition and categorization to train the machine by using convolutional neural networks (CNNs). There are a lot of models in CNN architectures such as VGG16, ResNet50, Xecption, and etc.

François (2018) based their study on clarification of drawing modules in convolutional neural networks which are the Xception model on an image classification dataset, there are focusing on the model performance by comparing Xception with another model.

In Kamath, Krupa and Thomas (2019) research focusing on classifier human picture by using 2 models (Xception and ResNet50) to classify also in the result, the F1-score of Xception models is higher than the other model. VGG16 is also one of the models that Swasono, Tjandrasa and Fathicah (2019) use to classification images as well and the result is satisfactory. Their research is using VGG16 to recognize the tobacco leaf and the result showed very high accuracy. On the other hand, they also have another architecture that will relate to this research. Rosebrock (2017) study about object detection and he uses another architecture that calls MobileNet in his image recognition and categorization and this architecture difference from CNN about MobileNet detects where is the target but CNN detects it in the image. So, all of the research that I mentioned before will be used in this research for image recognition and categorization architectures.

Conclusion

We show the experiment working process and our hypothesis after that, we prove our hypothesis and try different methods to reach our goal. This experiment has led us to improve the model to get the best accuracy. We presented a result in different methodologies (such as our own model, VGG16 and Xception) for the implementation part, comparing the result to get the best. In an experimental setting part, we try to solve the overfitting problem, class mode and Loss function, optimizer comparison, and batch size by using many methods and find the solution by comparing the accuracy of the model. In the end, we got the best model by using SGD as an optimizer, callback function, data augmentation, feature extraction by InceptionResnet and dropout as mentioned before in the result part.

A Proposal and Proof-of-Concept for Future work

Currently the project only uses images that were provided as part of the Stanford Dogs dataset, including the accompanying annotations data that holds the predefined bounding boxes. This is convenient for the purpose of training and testing accuracy, as it allows the background to be cropped away. Inconveniently, the real world does not come supplied with such bounding box data and as such any network being used in the real world will have to deal with this mostly useless background information.

As such, it would be very useful to remove that background in an automated fashion, such that we can focus on the objects we care about in a scene (dogs, in this case). We propose that this can be achieved using a secondary neural network at the pre-processing stage, prior to any images being given to the primary network. This network would be trained to detect dogs (breed is unimportant here) within an image and crop out those sections of the image without manual human intervention. Those crops can then be passed to the breed detecting network designed in this project, which would no longer have to concern itself with background data or performing its own cropping.

This sort of network already exists, as such, an example of this has been adapted as a proof of concept for this project, included in `autocropping.zip`. (The zipped Notebook contains deeper explanations on the code) It is based on an article written by A. Rosebrock (2017), mentioned in the literature review, using an existing MobileNet object detection network trained on the COCO (Common Objects in Context) dataset. This existing network is trained to detect 20 classes of objects, including dogs. By looping over every provided image we can use this network to pick out the dogs, get back coordinates of a bounding box, and perform very similar logic to our previous cropping code to return a new image.

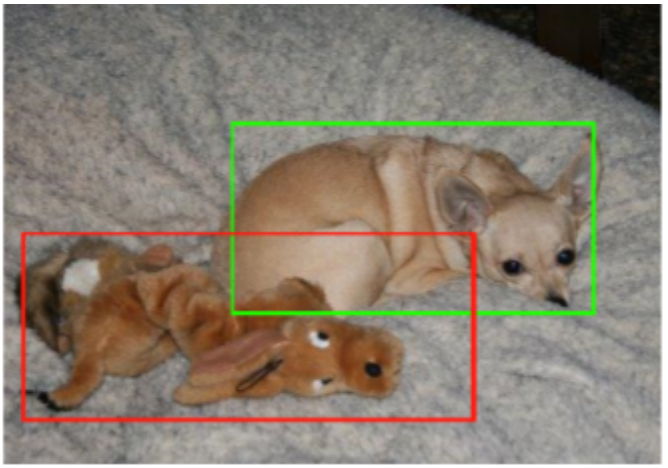

Figure 5: A good (green) and false-positive (red) detection by the existing object detection network

As it is, it’s slowed down by also detecting objects that are not dogs, and occasionally either misses the dog from an image entirely, or returns false positives, as seen in the figure to the right, but these are relatively rare and it is able to work through the entire dataset slightly faster than the previous code that relied upon the annotation data.

Issues aside, this concept shows promise of being highly efficient and accurate and a . it should be possible to improve it by either modifying the existing network with a method known as “Transfer Learning”, or by using the annotation data to train an entirely new network for this purpose.

References

Perez, L. and Wang, J., 2020. The Effectiveness Of Data Augmentation In Image Classification Using Deep Learning. [online] arXiv.org. Available at: <https://arxiv.org/abs/1712.04621> [Accessed 19 April 2020].

Keskar, N, S. and Saon, G., 2015. A nonmonotone learning rate strategy for SGD training of deep neural networks. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brisbane, QLD, 2015, pp. 4974-4978.

Masters, D. and Luschi, C., 2020. Revisiting Small Batch Training For Deep Neural Networks. [online] arXiv.org. Available at: <https://arxiv.org/abs/1804.07612> [Accessed 20 April 2020].

Ge, R., Huang, F., Jin, C. and Yuan, Y., 2020. Escaping From Saddle Points --- Online Stochastic Gradient For Tensor Decomposition. [online] arXiv.org. Available at: <https://arxiv.org/abs/1503.02101> [Accessed 20 April 2020].

François, C., 2018. Xception: Deep Learning with Depthwise Separable Convolutions. Computer Science - Computer Vision And Pattern Recognition,.

Kamath, V., Krupa, N. and Thomas, S., 2019. Hybrid Xception Model for Human Protein Atlas Image Classification. 2019 IEEE 16th India Council International Conference (INDICON), pp.1-4.

Rosebrock, A., 2017. Object Detection With Deep Learning And Opencv - Pyimagesearch. [online] PyImageSearch.Availableat:<https://www.pyimagesearch.com/2017/09/11/object-detection-with-deep-learning-and-opencv/>.

Swasono, D., Tjandrasa, H. and Fathicah, C., 2019. Classification of Tobacco Leaf Pests Using VGG16 Transfer Learning. 2th International Conference on Information & Communication Technology and System (ICTS), pp.176-181.

P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. IEEE Transactions on Pattern Analysis and Machine Intelligence, 32(9):1627–1645, 2010